secchan

Srini, rs@toprllc.com

August 25, 2021

1 Introduction

This projectlet is an extension of the password generator whereby the generator resides inside a server and the client simply requests a password.

The exchanges are encrypted with a simple key exchange scheme.

2 Secure Channel Design

2.1 Libraries

A secure file facility and in particular a secure password file is designed to support this tool. Encryption, key derivation and other such building blocks are implemented using **libsodium** libraries (`https://doc.libsodium.org/`).

2.2 Public Key sealed boxes

This implementation utilizes the public key sealed boxes as the technique (`https://doc.libsodium.org/public-key_cryptography/sealed_boxes`). Basic idea in this method is that the sender of a message encrypts a message with the public key of the receiver. Upon receipt of the encrypted message, the receiver can decrypt the message using the receiver's secret key.

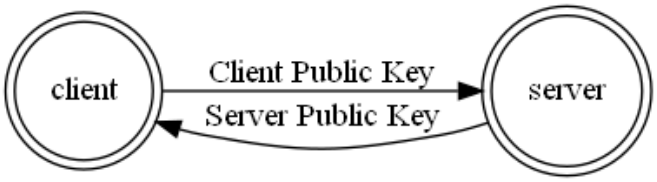
Thus the channel setup begins with the exchange of the public keys.

2.3 Message Format

In order to support payloads of varying sizes, the messages start with the payload size field. Following the size is the payload encrypted with the partner's public key.

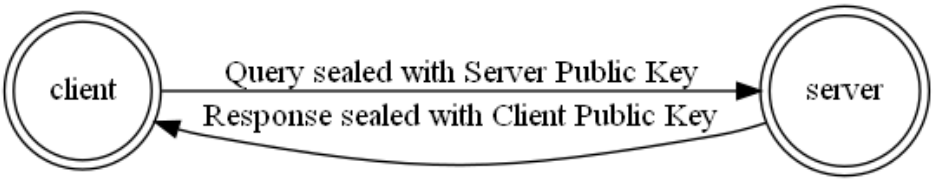
2.4 Session Setup

Session setup involves the exchange of public keys. Upon connecting a stream socket, the client sends its public key as the first message. The expected response for this from the server is the server's public key as illustrated:



2.5 Transactions

Once the channel is setup, transactions can take place. In this library, the transactions begin with a query from the client always resulting in a response from the server. Other patterns may well be implemented as long as the payloads are **seal**ed with the partner's public key.



3 Example Application

The secure channel is used to develop a password server. Clients can query the server for a password and the server returns a password with the specified characteristics ie number of segments and the segment separator.

Individual messages in the exchange are simplistic but of varying sizes illustrating the applicability of the secure channel.

The example application operates as a server or a client.

3.1 Usage

```
$ obj/secpwdgen.exe --help
secpwdgen V01 2021-08-23 04:34:16
Usage: secpwdgen.exe secpwdgen

 -v, --verbose                Output extra verbose information
 -c, --client                 client option.
 -s, --segments=ARG           Number of Segments
 -r, --separator=ARG          Separator - default: -
```

```
 -t, --target-host=ARG         Target Host - default: 127.0.0.1
 -p, --port=ARG                Target Port
 -n, --number-of-passwords=ARG Number of Passwords
```

3.2 Client operation

The client is invoked with the default host of **127.0.0.1**. Each invocation specifies different password parameters eg number of segments, separators.

```
$ obj/secpwdgen.exe --client -p=1024
cataract-8786-Fluid-36229

$ obj/secpwdgen.exe --client -p=1024 -r=^ -s=1
eye^7566

$ obj/secpwdgen.exe --client -p=1024 -r=? -s=3
revati?8618?Brain?34376?spectrum?37648

$ obj/secpwdgen.exe --client -p=1024 -r=? -s=3 -n=4
hogweed?18793?Radar?27708?sita?20703
occam?47114?Anatomy?42467?arete?24505
cornea?16379?Synonychia?40886?vein?21029
hindolam?38136?Raman?34325?arabi?23021
```

3.3 Server operation

The server is started up on port 1024.

```
$ obj/secpwdgen.exe -p=1024
Server No  1
Server No  2
Server No  3
Server No  4
Server No  5
Server No  6
Server No  7
```

4 Implementation

Ada binding to the **libsodium** library is developed as part of this effort:

```
https://gitlab.com/ada23/sodiumada.git
```

The secure channel facility and an application to generate passwords by a remote server can be found in:

```
https://gitlab.com/ada23/secchan.git
```